
University of Southern California
Center for SoftwareEngineering

CeBASE

# High Dependability Computing in a Competitive World

Barry Boehm, USC

IEEE-NASA SW Engineering Workshop

November 28, 2001

(boehm@; http://) sunset.usc.edu

# HDC in a Competitive World

- The economics of IT competition and dependability
- Software Dependability Opportunity Tree
  - Decreasing defects
  - Decreasing defect impact
  - Continuous improvement
- Conclusions and References

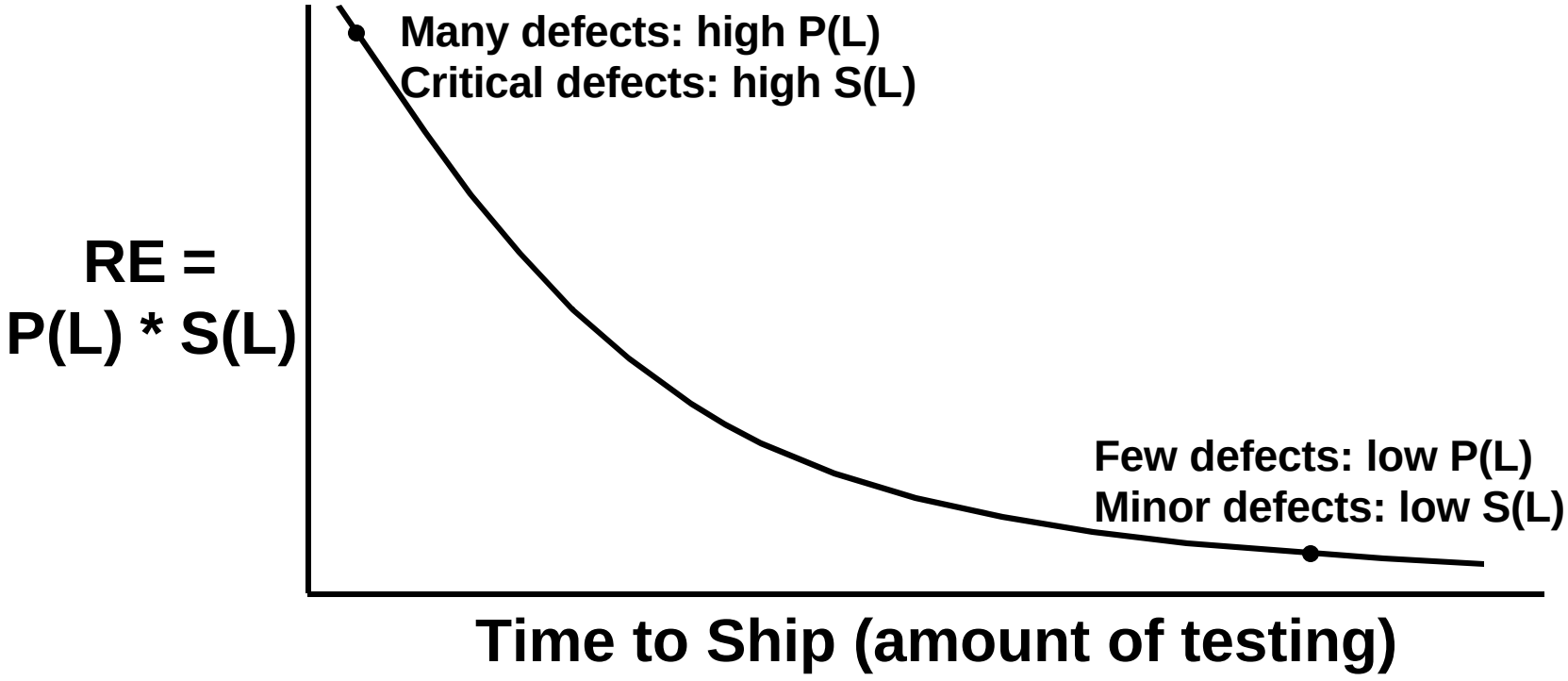
# Competing on Schedule and Quality
## - A risk analysis approach

- Risk Exposure RE = Prob (Loss) * Size (Loss)
  - "Loss" – financial; reputation; future prospects, …
- For multiple sources of loss:

$$RE = \sum_{sources} [Prob(Loss) * Size(Loss)]_{source}$$

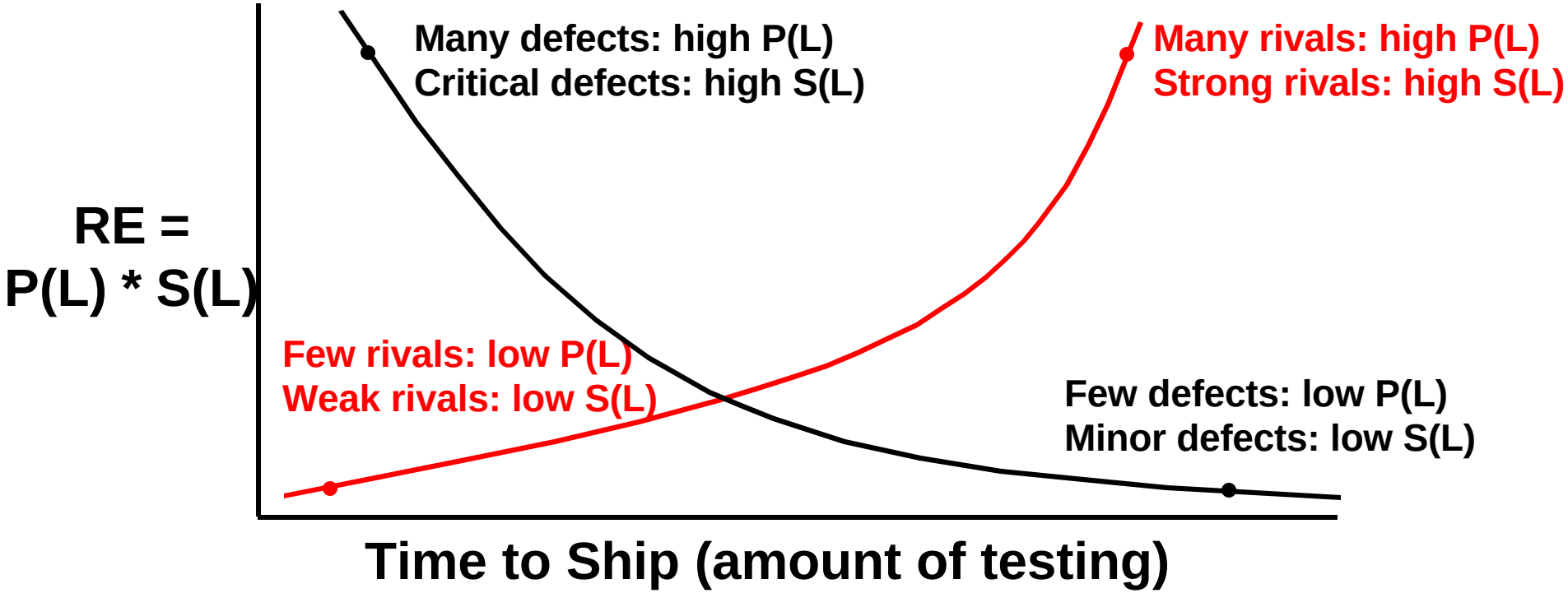
# Example RE Profile: Time to Ship

## - Loss due to unacceptable dependability



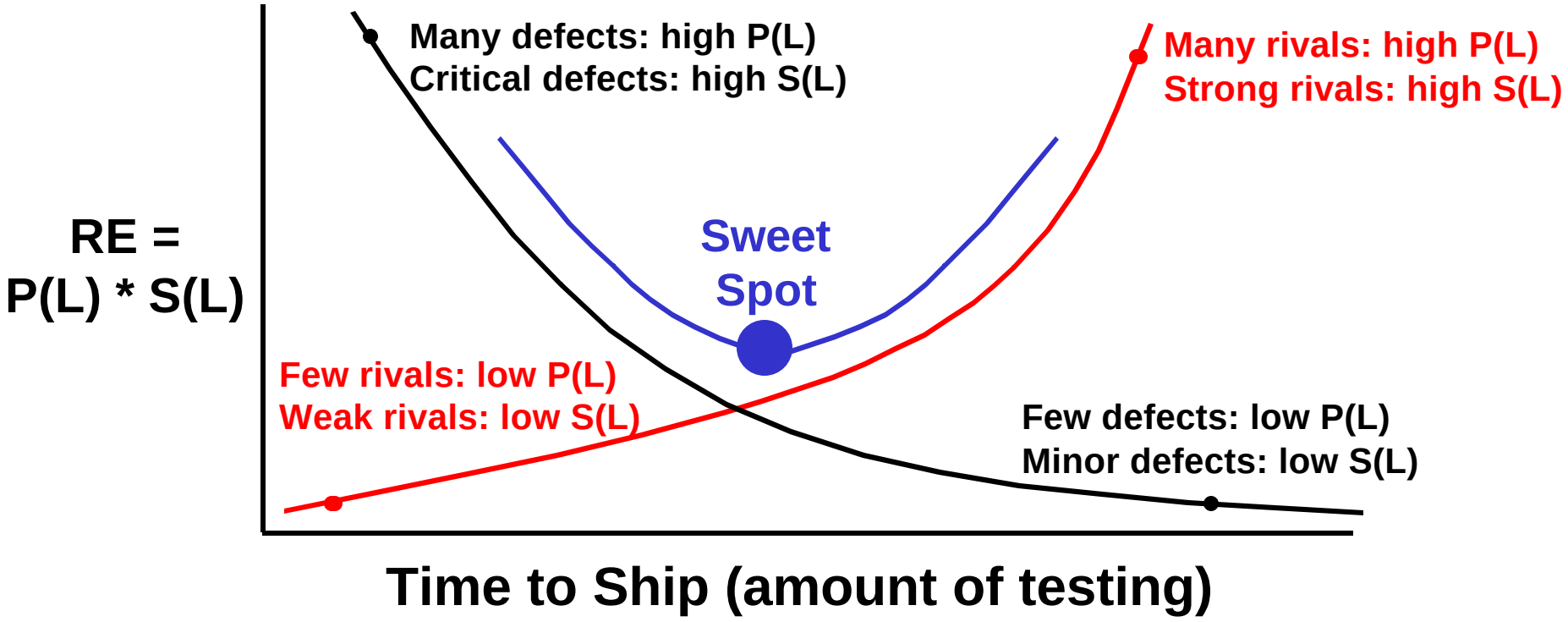
# Example RE Profile: Time to Ship

- Loss due to unacceptable dependability
  - Loss due to market share erosion



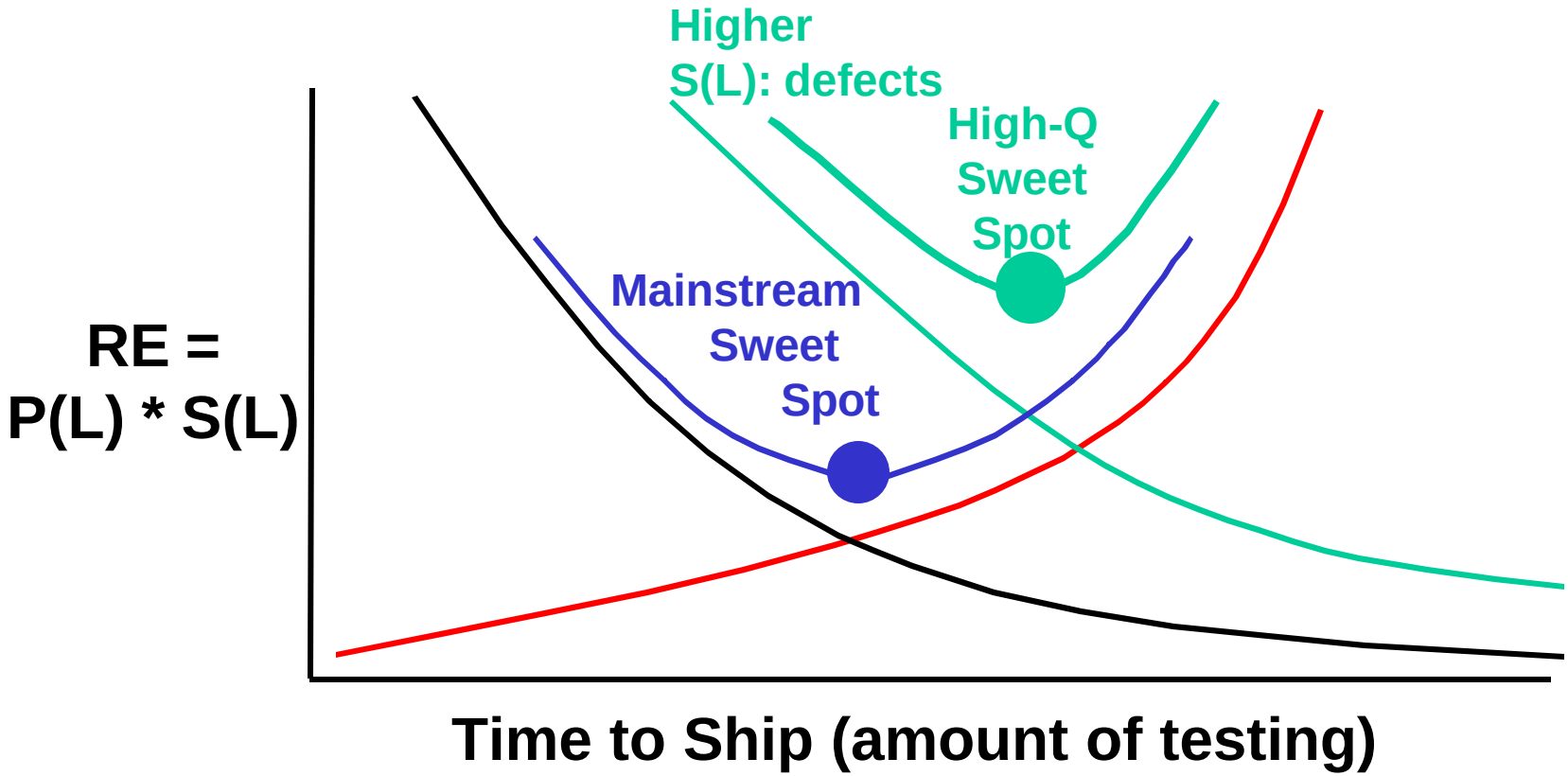
# Example RE Profile: Time to Ship

## - Sum of Risk Exposures

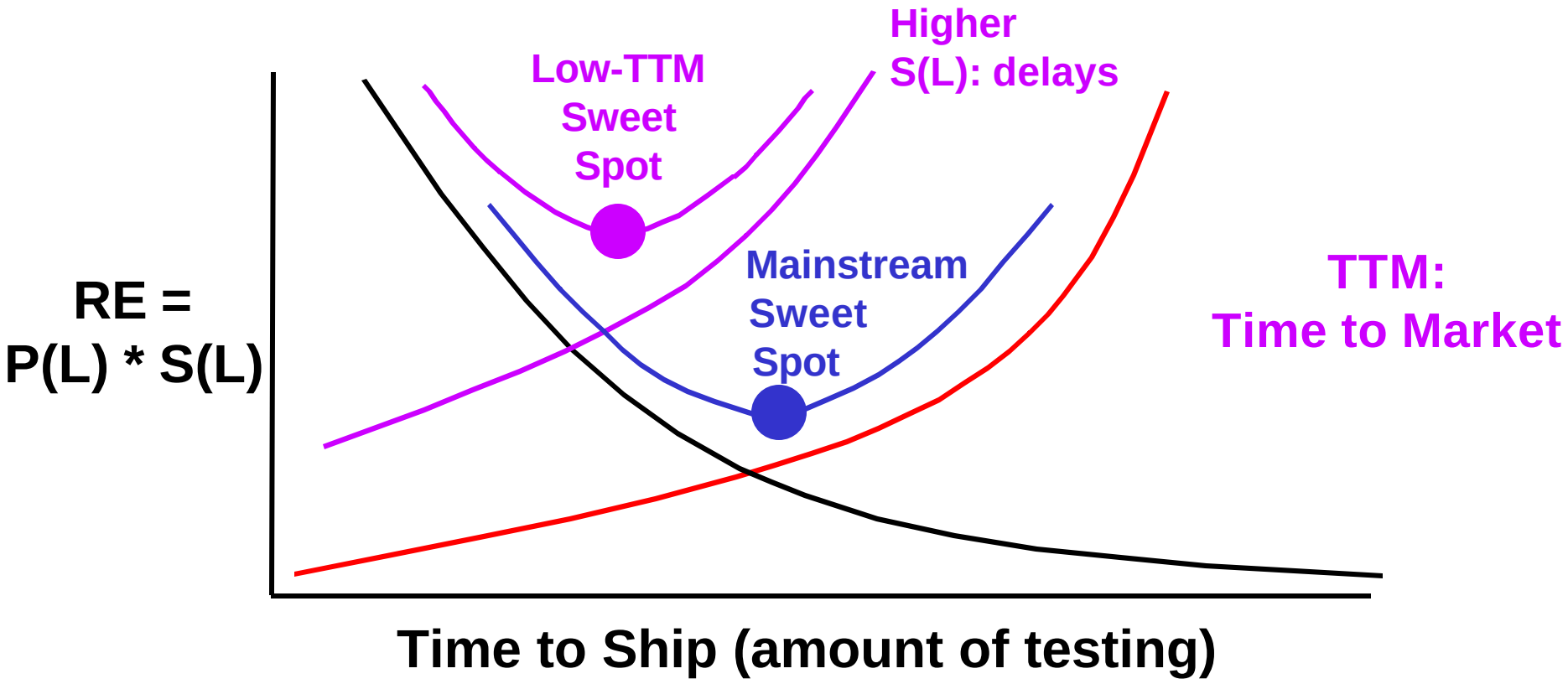


# Comparative RE Profile:

## Safety-Critical System



# Comparative RE Profile: Internet Startup

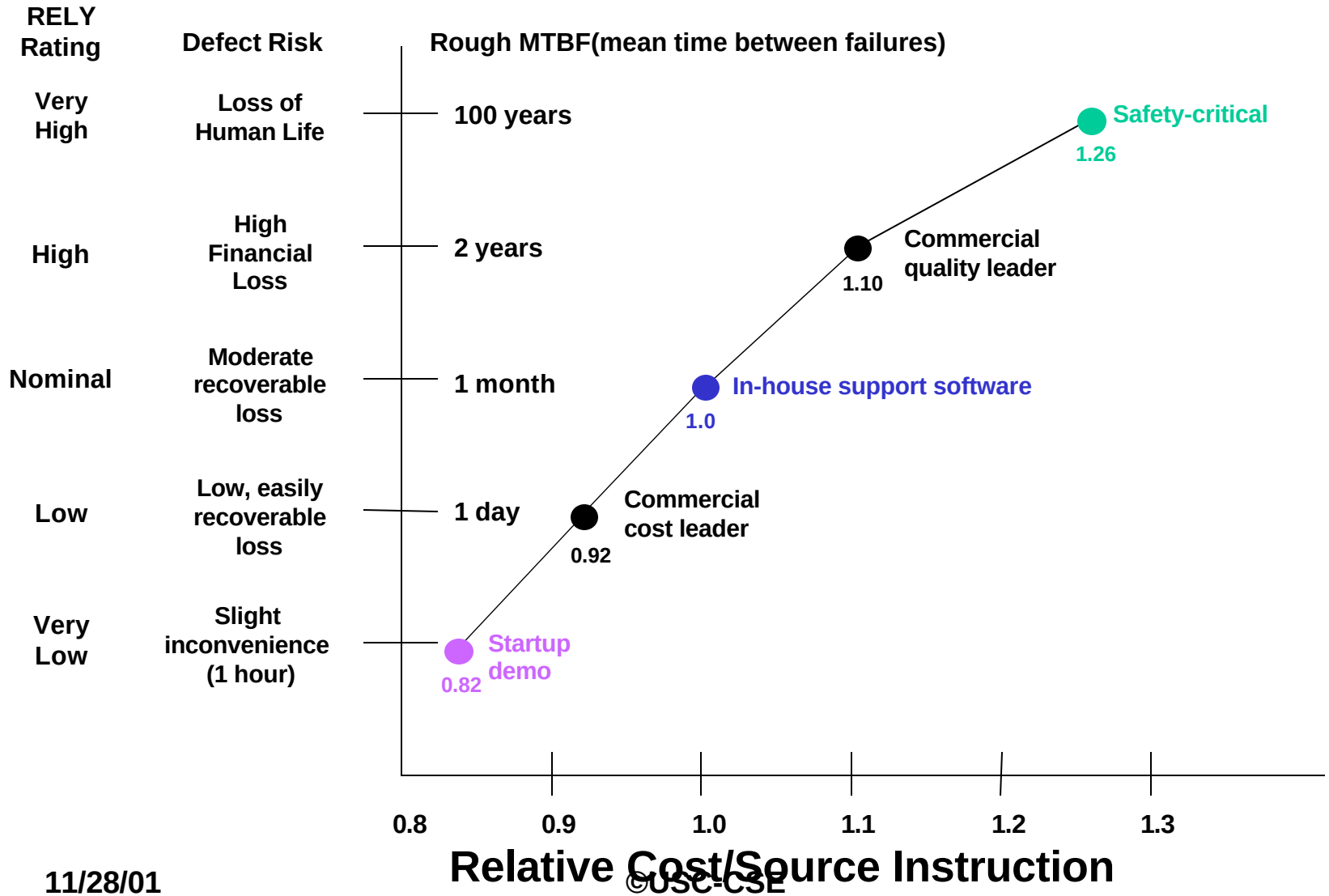


# Conclusions So Far

- Unwise to try to compete on both cost/schedule and quality
  - Some exceptions: major technology or marketplace edge
- There are no one-size-fits-all cost/schedule/quality strategies
- Risk analysis helps determine how much testing (prototyping, formal verification, etc.) is enough
  - Buying information to reduce risk
- Often difficult to determine parameter values
  - Some COCOMO II values discussed next

# Software Development Cost/Quality Tradeoff

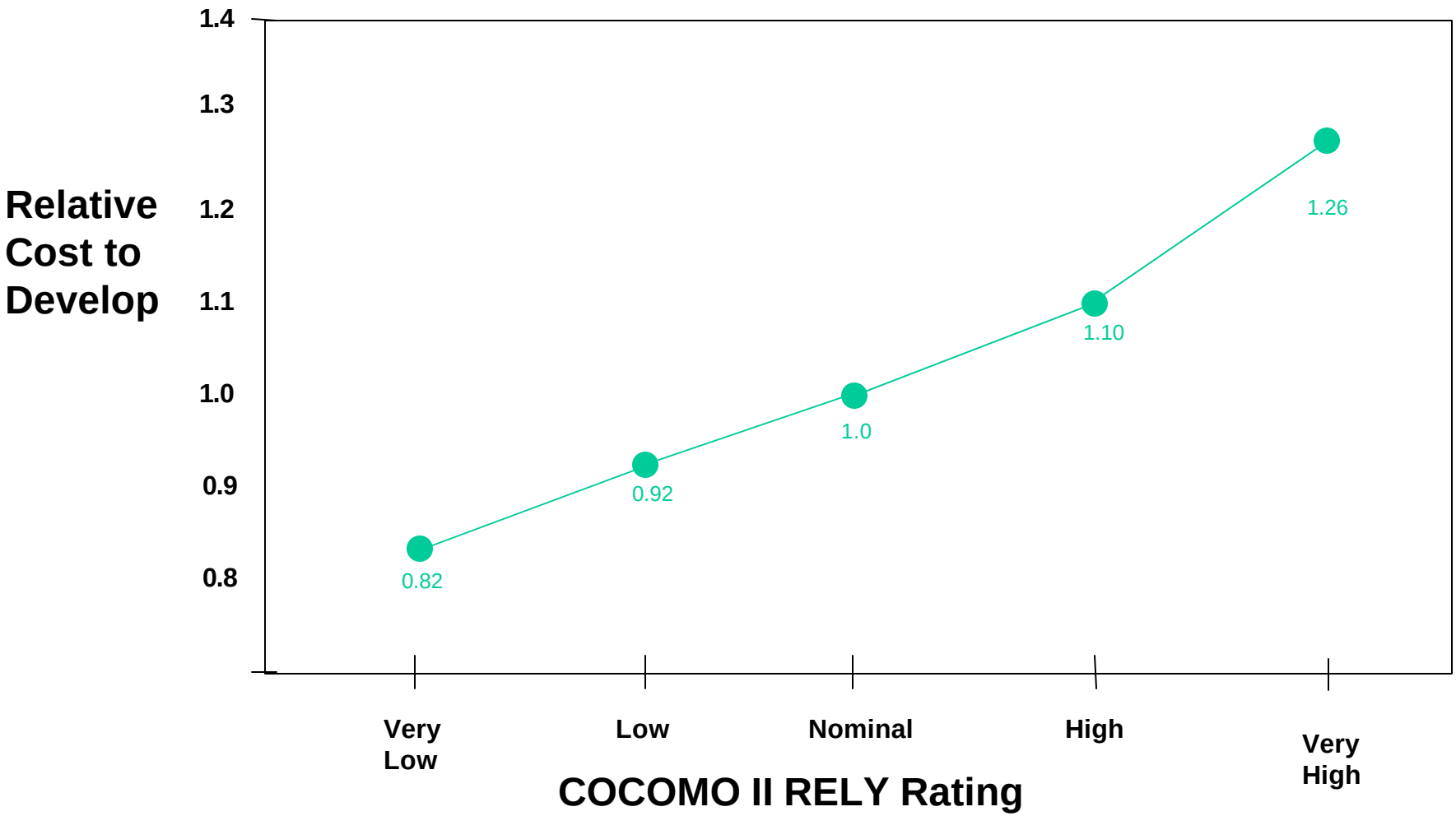
## - COCOMO II calibration to 161 projects



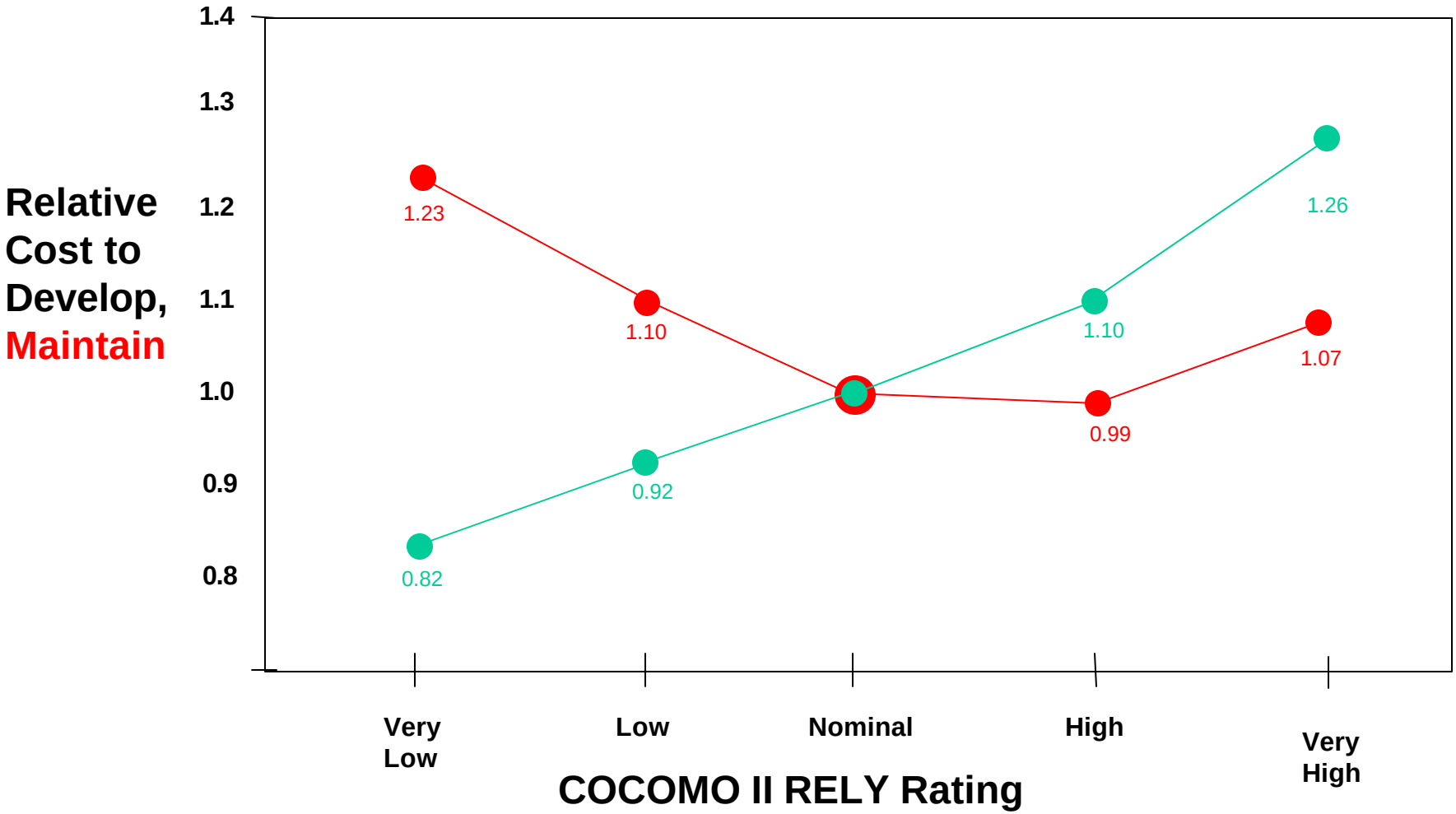
# "Quality is Free"

- Did Philip Crosby's book get it all wrong?
- Investments in dependable systems
  - Cost extra for simple, short-life systems
  - Pay off for high-value, long-life systems

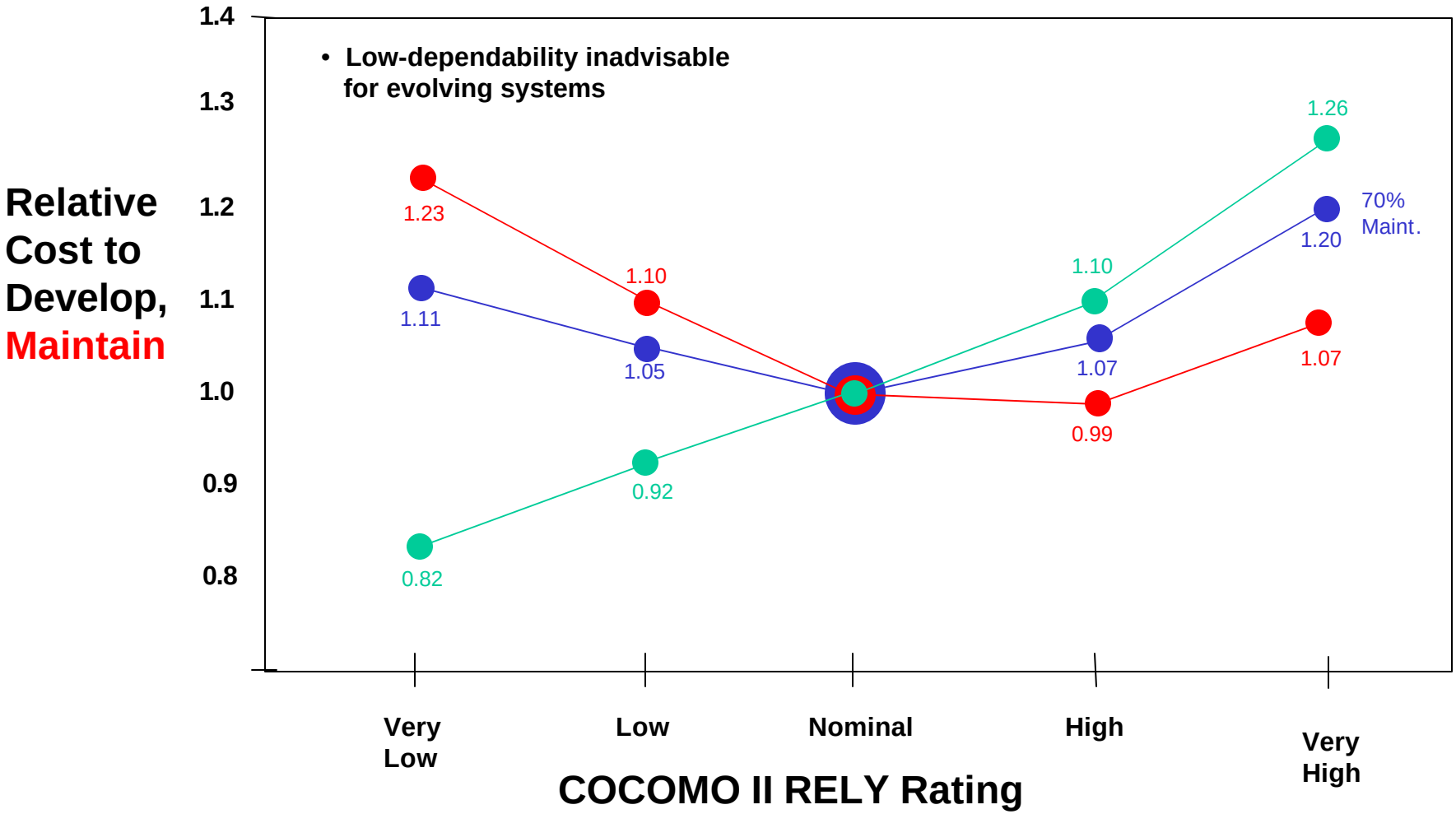
# Software Life-Cycle Cost vs. Dependability



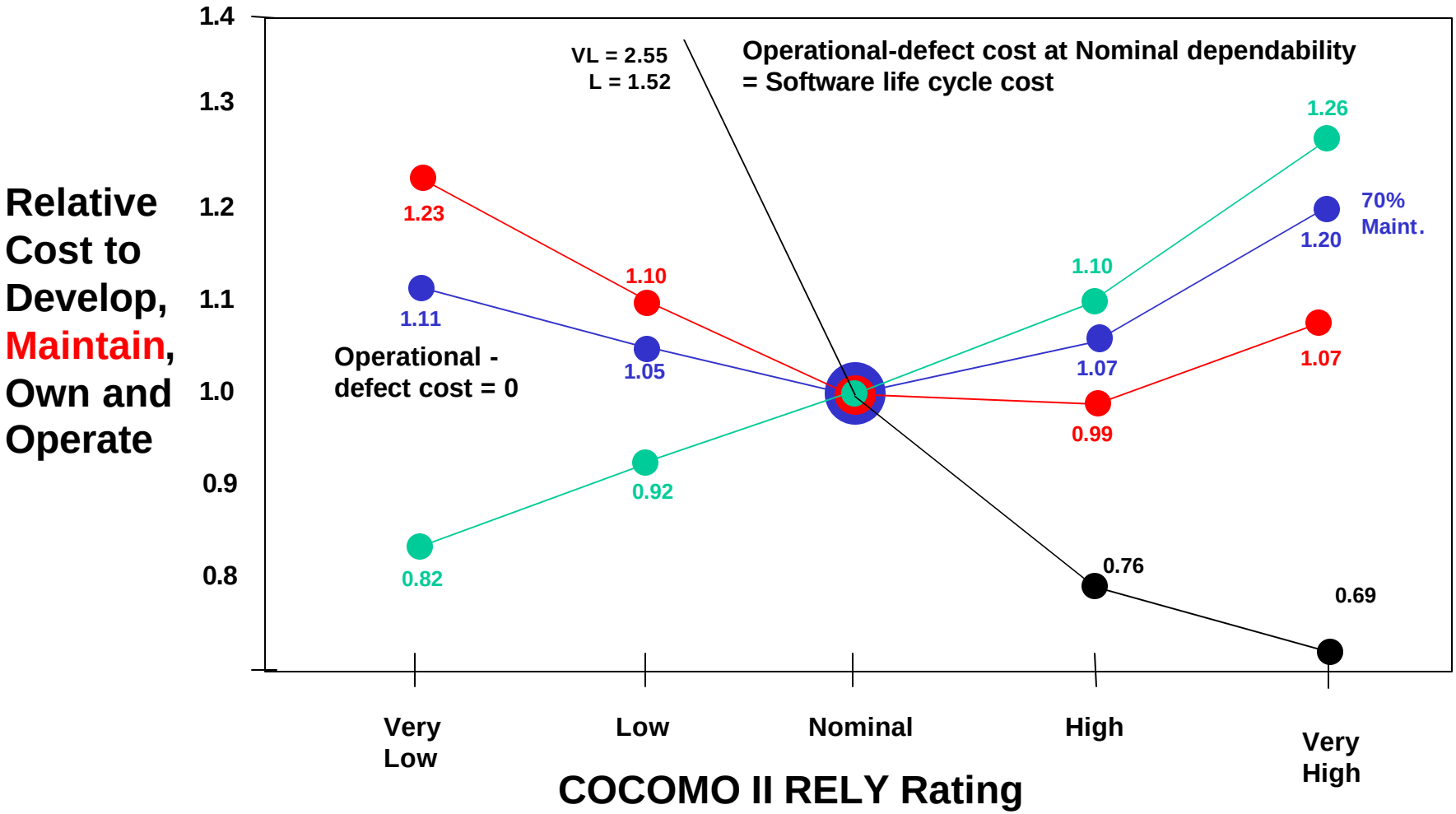
# Software Life-Cycle Cost vs. Dependability



# Software Life-Cycle Cost vs. Dependability



# Software Ownership Cost vs. Dependability



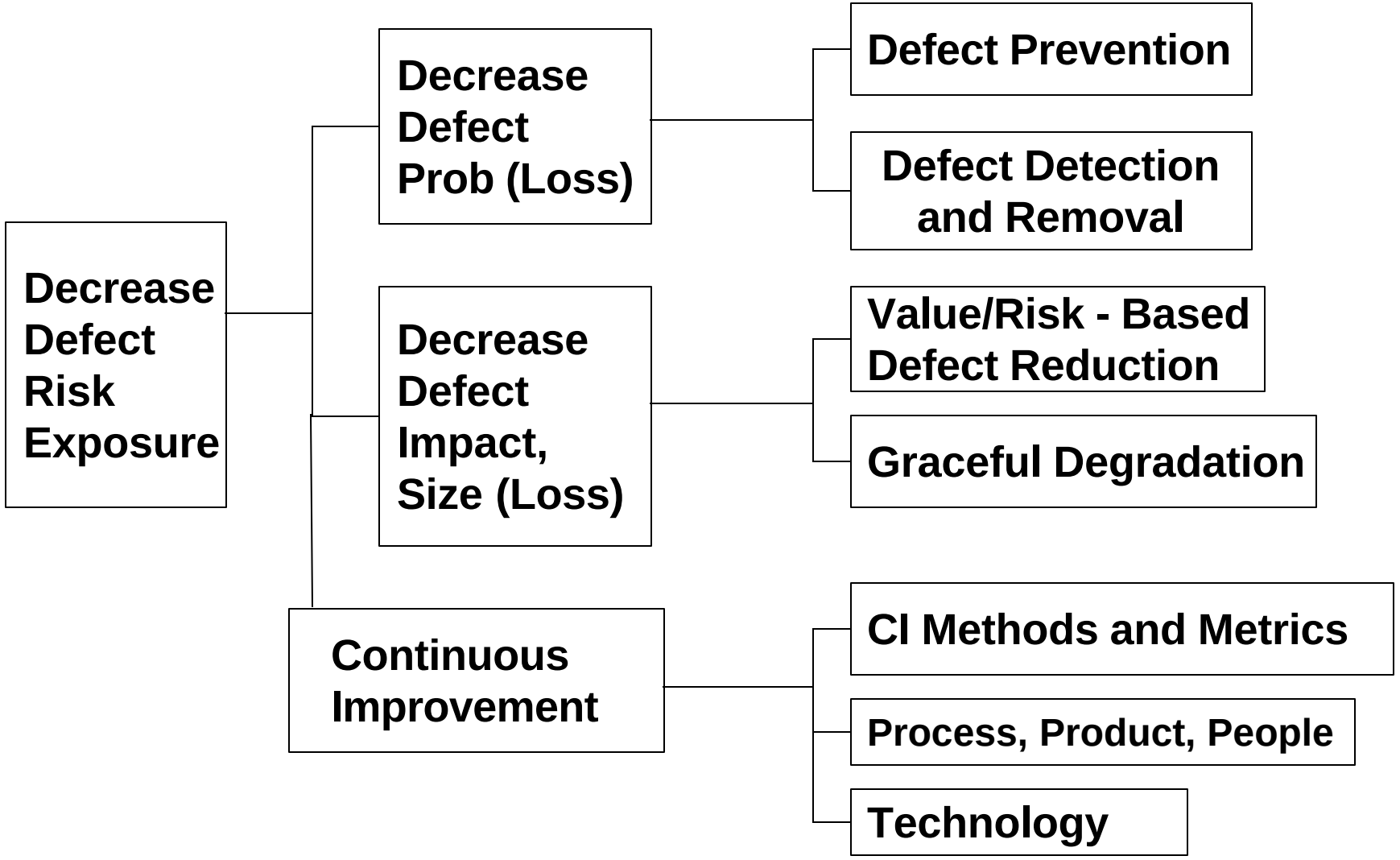
# Conclusions So Far - 2

- Quality is better than free for high-value, long-life systems
- There is no universal dependability sweet spot
  - Yours will be determined by your value model
  - And the relative contributions of dependability techniques
    - Let's look at these next

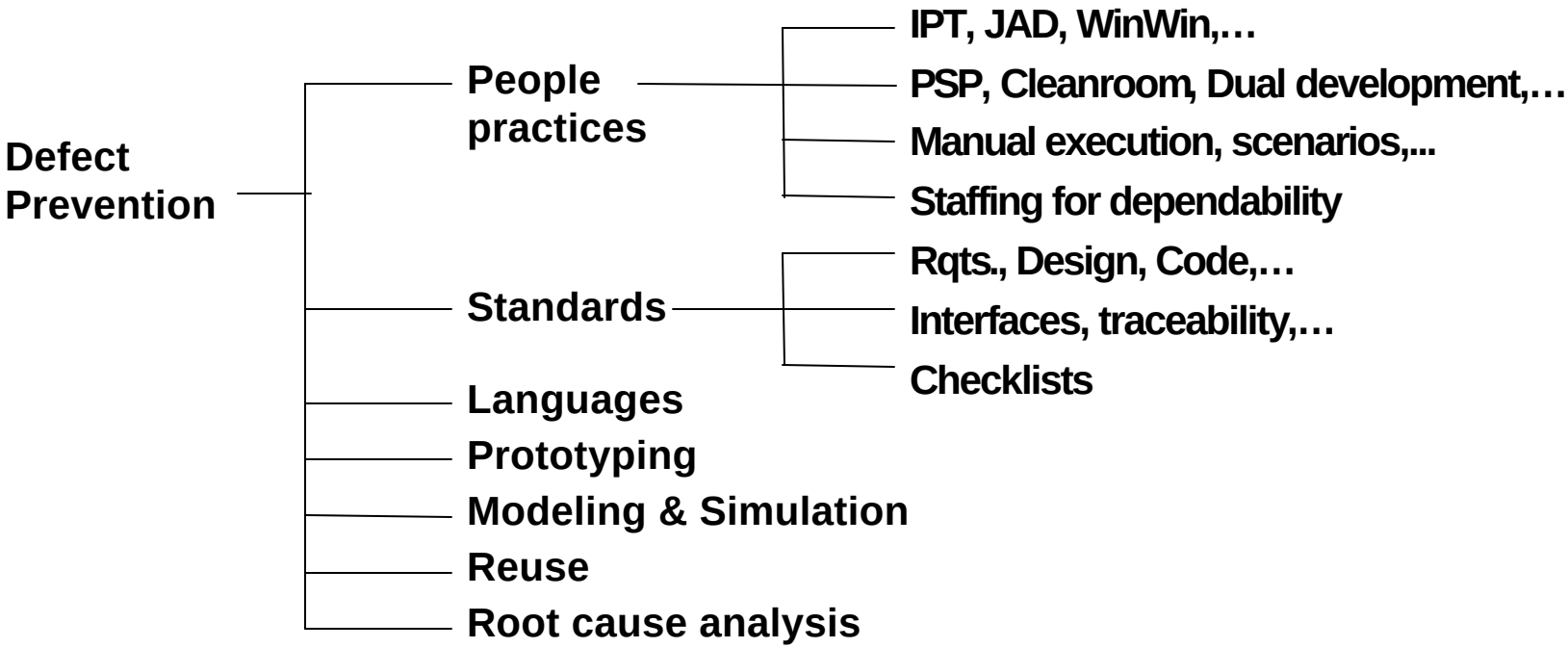
# HDC in a Competitive World

- The economics of IT competition and dependability
- Software Dependability Opportunity Tree
  - Decreasing defects
  - Decreasing defect impact
  - Continuous improvement
- Conclusions

# Software Dependability Opportunity Tree



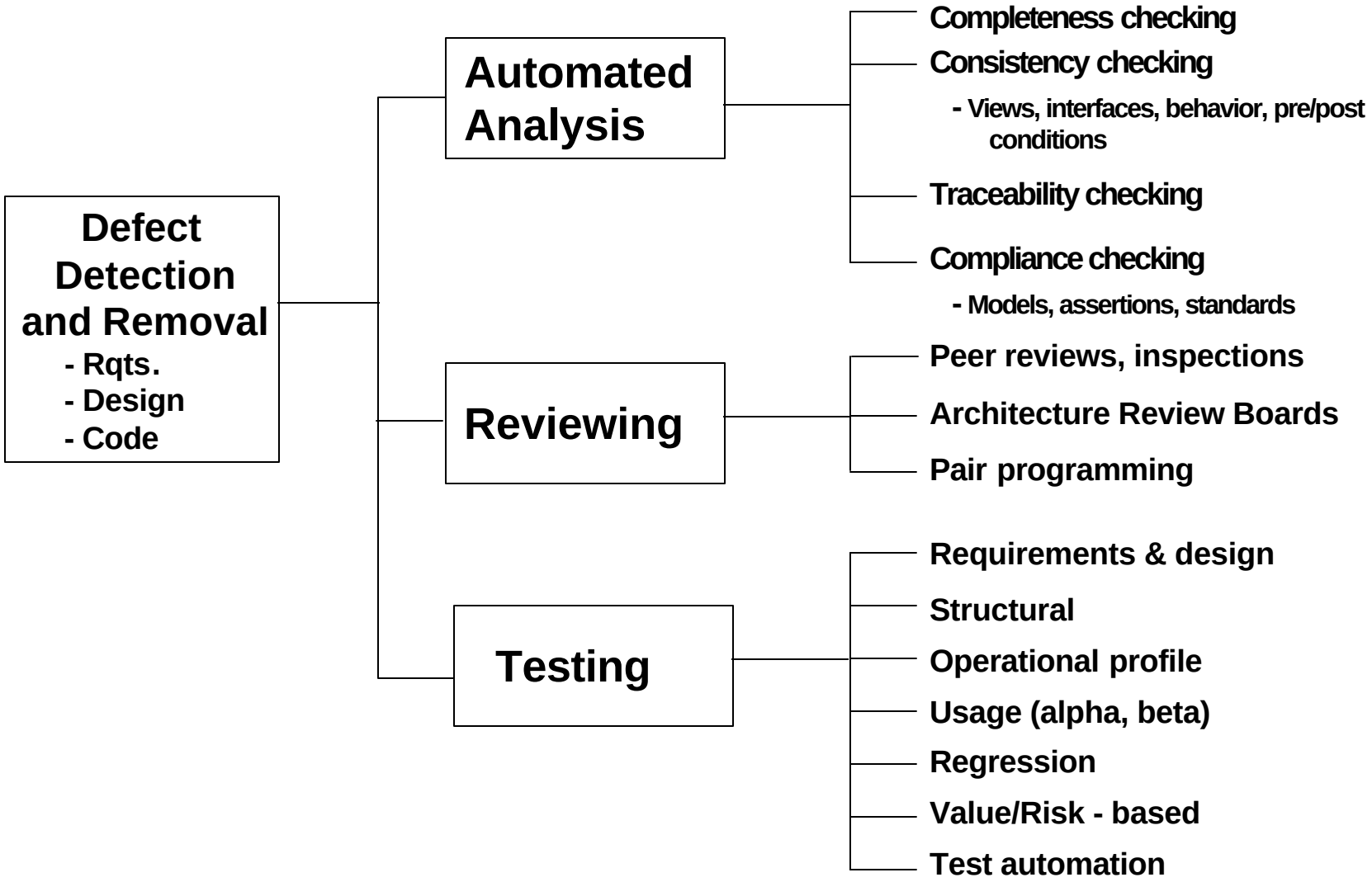
# Software Defect Prevention Opportunity Tree



# People Practices: Some Empirical Data

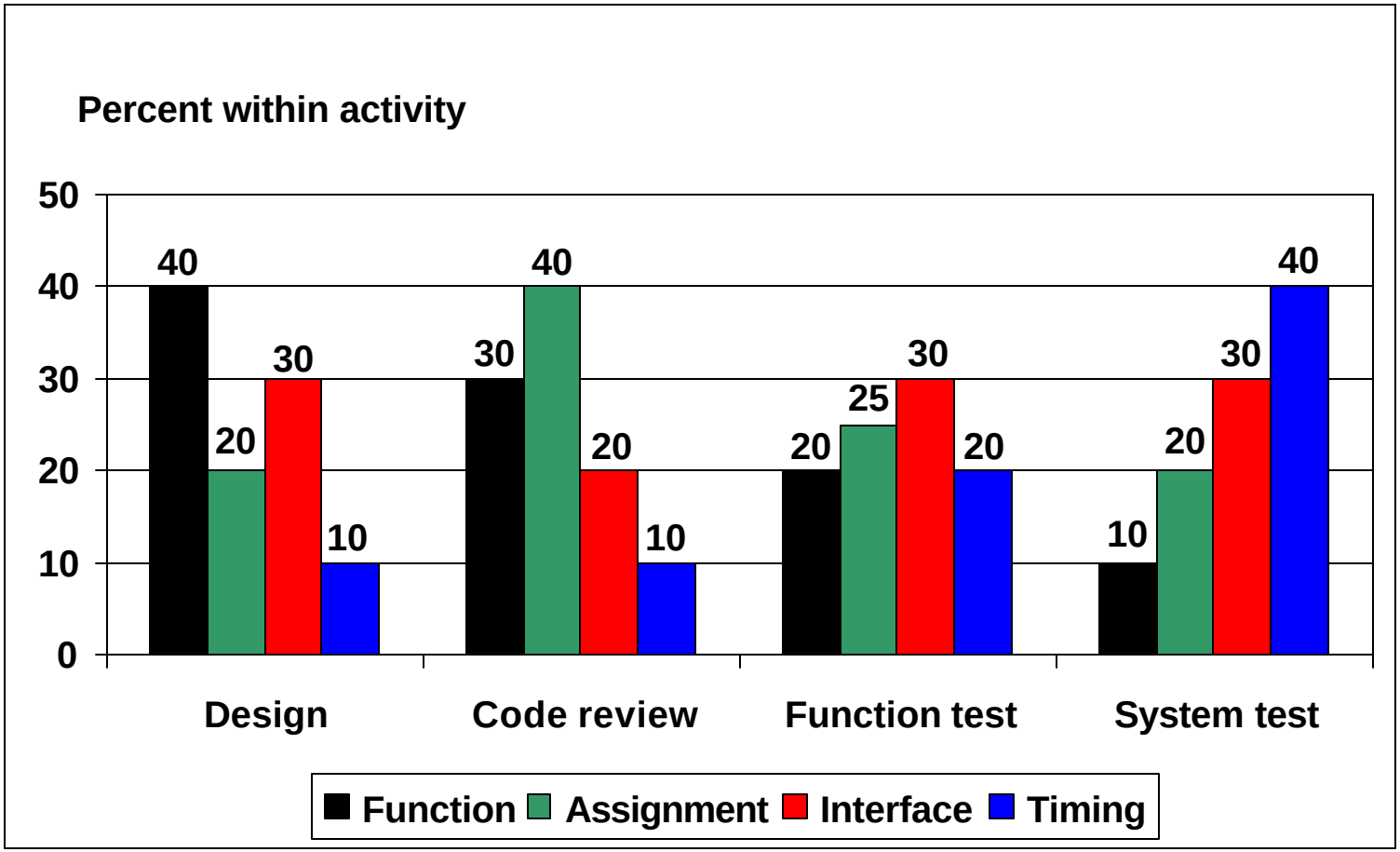
- Cleanroom: Software Engineering Lab
  - 25-75% reduction in failure rates
  - 5% vs 60% of fix efforts over 1 hour
- Personal Software Process/Team Software Process
  - 50-75% defect reduction in CMM Level 5 organization
  - Even higher reductions for less mature organizations
- Staffing
  - Many experiments find factor-of-10 differences in people's defect rates

# Software Defect Detection Opportunity Tree

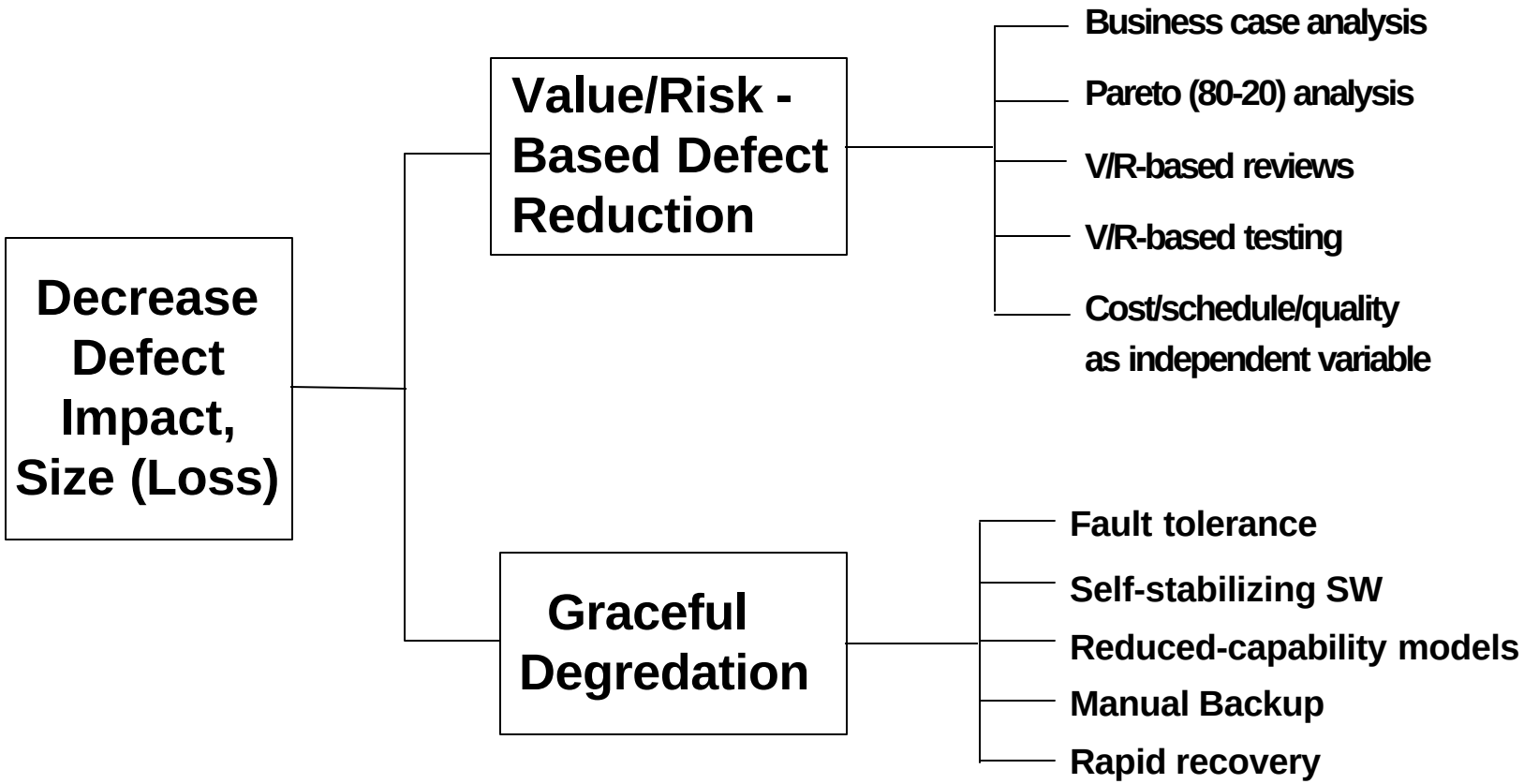


# Orthogonal Defect Classification

## - Chillarege, 1996



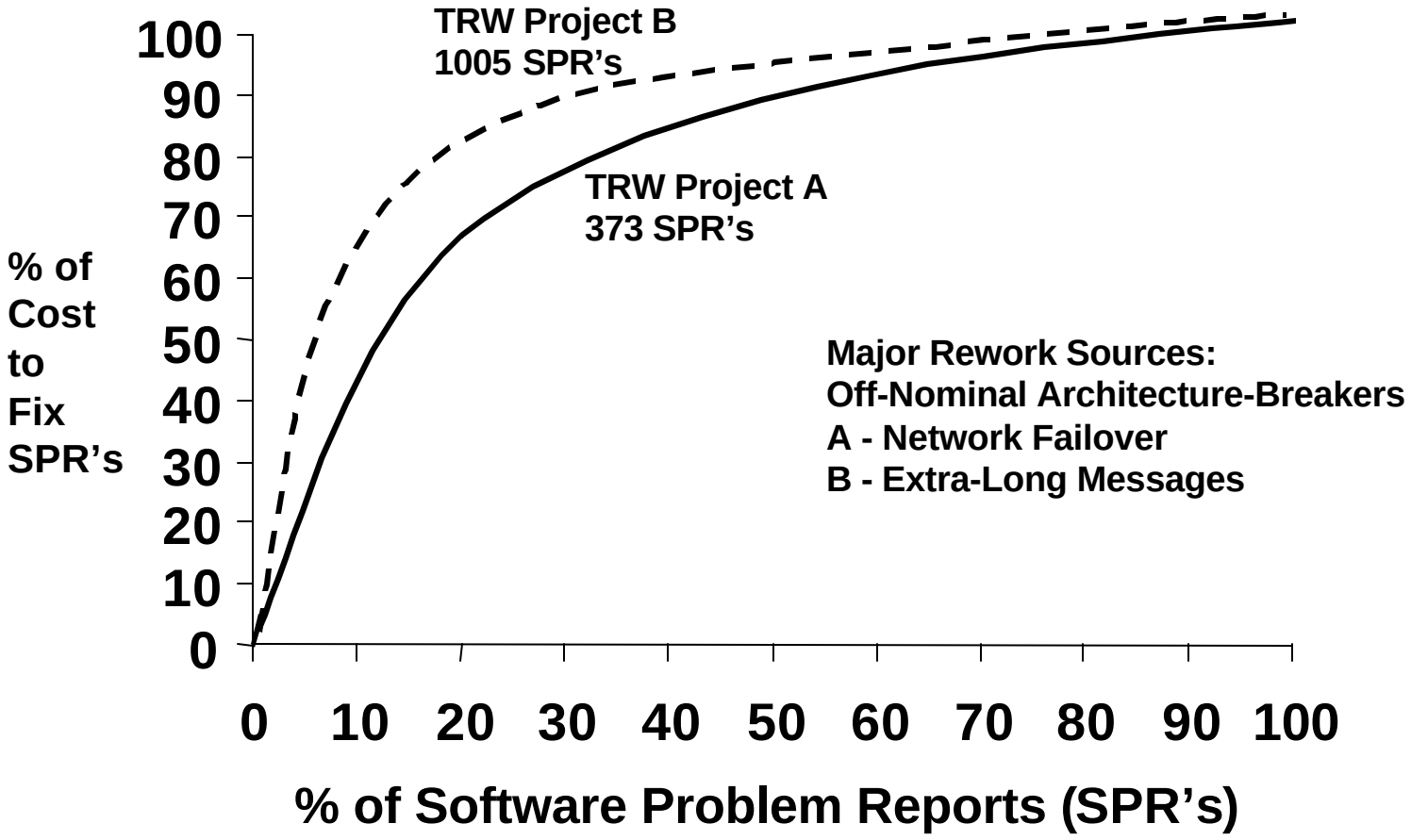
# Defect Impact Reduction Opportunity Tree



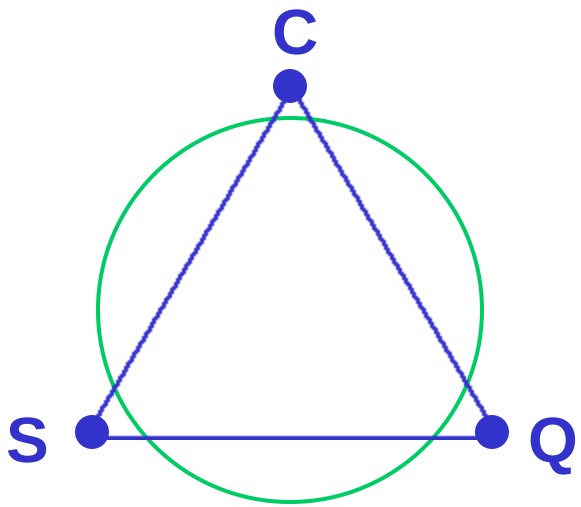
# Pareto 80-20 Phenomena

- 80% of the rework comes from 20% of the defects
- 80% of the defects come from 20% of the modules
  - About half the modules are defect-free
- 90% of the downtime comes from < 10% of the defects

# Pareto Analysis of Rework Costs

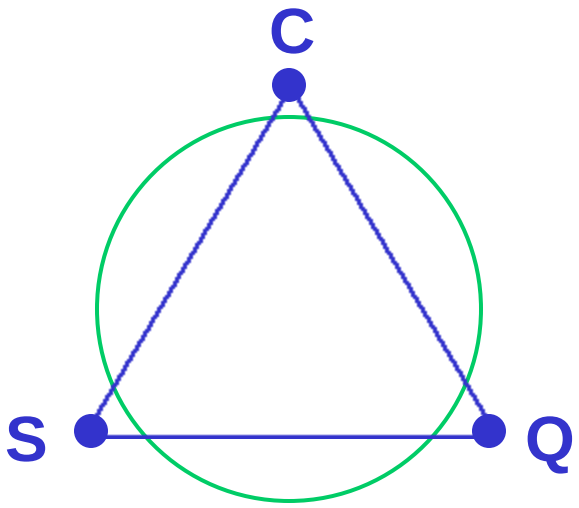


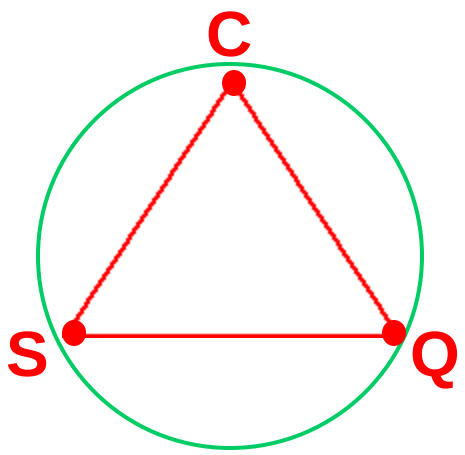
# Cost, Schedule, Quality: Pick any Two?



# Cost, Schedule, Quality: Pick any Two?

- Consider C, S, Q as Independent Variable
  - Feature Set as Dependent Variable



# C, S, Q as Independent Variable

- Determine Desired Delivered Defect Density (D4)
  - Or a value-based equivalent
- Prioritize desired features
  - Via QFD, IPT, stakeholder win-win
- Determine Core Capability
  - 90% confidence of D4 within cost and schedule
  - Balance parametric models and expert judgment
- Architect for ease of adding next-priority features
  - Hide sources of change within modules (Parnas)
- Develop core capability to D4 quality level
  - Usually in less than available cost and schedule
- Add next priority features as resources permit
- Versions used successfully on 17 of 19 USC digital library projects

# Conclusions

- Future trends intensify competitive HDC challenges
  - Complexity, criticality, decreased control, faster change
- Organizations need tailored, mixed HDC strategies
  - No universal HDC sweet spot
  - Goal/value/risk analysis useful
  - Quantitative data and models becoming available
- HDC Opportunity Tree helps sort out mixed strategies
- Quality is better than free for high-value, long-life systems
- Attractive new HDC technology prospects emerging
  - Architecture- and model-based methods
  - Lightweight formal methods
  - Self-stabilizing software
  - Complementary theory and empirical methods

# References

V. Basili et al., "SEL's Software Process Improvement Program," IEEE Software, November 1995, pp. 83-87.

B. Boehm and V. Basili, "Software Defect Reduction Top 10 List," IEEE Computer, January 2001

B. Boehm et al., Software Cost Estimation with COCOMO II, Prentice Hall, 2000.

J. Bullock, "Calculating the Value of Testing," Software Testing and Quality Engineering, May/June 2000, pp. 56-62

CeBASE (Center for Empirically-Based Software Engineering), http://www.cebase.org

R. Chillarege, "Orthogonal Defect Classification," in M. Lyu (ed.), Handbook of Software Reliability Engineering, IEEE-CS Press, 1996, pp. 359-400.

P. Crosby, Quality is Free, Mentor, 1980.

R. Grady, Practical Software Metrics, Prentice Hall, 1992

N. Leveson, Safeware: System Safety and Computers, Addison Wesley, 1995

B. Littlewood et al., "Modeling the Effects of Combining Diverse Fault Detection Techniques," IEEE Trans. SW Engr. December 2000, pp. 1157-1167.

M. Lyu (ed), Handbook of Software Reliability Engineering, IEEE-CS Press, 1996

J. Musa and J. Muda, Software Reliability Engineered Testing, McGraw-Hill, 1998

M. Porter, Competitive Strategy, Free Press, 1980.

P. Rook (ed.), Software Reliability Handbook, Elsevier Applied Science, 1990

W. E. Royce, Software Project Management, Addison Wesley, 1998.

# CeBASE Software Defect Reduction Top-10 List - http://www.cebase.org

1. Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase.
2. About 40-50% of the effort on current software projects is spent on avoidable rework.
3. About 80% of the avoidable rework comes form 20% of the defects.
4. About 80% of the defects come from 20% of the modules and about half the modules are defect free.
5. About 90% of the downtime comes from at most 10% of the defects.
6. Peer reviews catch 60% of the defects.
7. Perspective-based reviews catch 35% more defects than non-directed reviews.
8. Disciplined personal practices can reduce defect introduction rates by up to 75%.
9. All other things being equal, it costs 50% more per source instruction to develop high-dependability software products than to develop low-dependability software products. However, the investment is more than worth it if significant operations and maintenance costs are involved.
10. About 40-50% of user programs have nontrivial defects.